Assignment 4-K means

Siwei Li Siqi Wang Ruhan Zhang

October 30, 2015

Part 3: K means

Before starting this part, we first load two packages: fpc and flexclust. The package fpc will allow us to use the command plotcluster() to visualize the clustering result. The package flexclust will allow us to use the function randIndex() to evaluate the clustering result.

```
library('fpc')
library('flexclust')
```

```
## Loading required package: grid
## Loading required package: lattice
## Loading required package: modeltools
## Loading required package: stats4
```

Instead of using the built-in function kmeans() directly, we try to write a R function for K means algorithm. The function is called clusterByKmeans() with two parameters: the data set and the number of groups. We closely follow the K-means algorithm to write this function. The logic of the algorithm as well as the function clusterByKmeans() is as follows.

Suppose we want to cluster all the data points into k groups. Step 1: Randomly pick k data points as the initial centers (means) of the k groups. Step 2: Assign each data point to the group whose center is closest to to this data point in terms of Euclidean distance. Each point is assigned to only one group. This step will then gives k groups of data points. Step 3: Calculate the new centers (means) of the k groups and update the values of the centers (means) of the k groups. Step 4: Repeat Step 2 and Step 3, until the assignment of all data points does not change any more.

The code of the function clusterByKmeans is given below:

```
clusterByKmeans <- function(dataset, groupNo){

# Obtain the number of rows and the number of columns of the dataset
rowNo <- nrow(dataset)
colNo <- ncol(dataset)

# Create a vector to store the clustering result
# (indicating which point belongs to which cluster)
cluster <- vector("numeric", length=rowNo)

# Create a vector to store the distance between each data point
# and the center of the cluster to which that data point belongs.
dtc <- vector("numeric", length=rowNo)

# Create a matrix to store the centers of the groups
# Use a matrix because each center is a row vector
center <- matrix(0,nrow=groupNo,ncol=colNo)

# Randomly select groupNo points (rows of dataset)
```

```
# as the initial centers of the groupNo groups
randomInt <- as.vector(sample(1:rowNo,size=groupNo))
for (i in 1:groupNo){
    center[i,] <- dataset[randomInt[i],]
    center <- matrix(center,groupNo,colNo)
}

# We specify an intial way of clustering:
# all the rest of the points belong to the first cluster,
# while the points chosen as the initial centers belong to its own cluster.
for (i in 1:rowNo){
    cluster[i] <- 1
}
for (j in 1:groupNo){
    cluster[randomInt[j]] <- j
}


# Calculate the initial value of the distance vector, dtc
for (i in 1:rowNo){
    dtc[i] <- sqrt(sum((dataset[i,]-center[cluster[i],])^2))
}

# Create a variable, changed, which indicates
# whether the clustering result is changed in an iteration.
changed <- TRUE

# If the assignment is changed in a iteration, we continue;
# if the assignment no longer changes, we stop.
while(changed==TRUE){

initialCluster <- cluster

for (i in 1:rowNo){
    initialdtc <- dtc[i]
    preCluster <- cluster[i]

    # We shall find a better cluster solution for the point i
    # in terms of smaller Euclidean distance.
    for (j in 1:groupNo){
        # Compute the distance between the point i and the center of the cluster j
        currentdtc <- sqrt(sum((dataset[i,]-center[j,])^2))
        if (currentdtc<initialdtc){
            # Update the clustering result
            cluster[i] <- j
            # Update the distance vector
            dtc[i] <- currentdtc
        }
    }
  }

# Compare the new assignment with the previous assignment.
# The overall assignment is considered as "changed" if the assignment of
```

```
# at least one point is changed.
for (i in 1:rowNo){
    if (cluster[i] != initialCluster[i]){
        changed <- TRUE
        break
    }else{
        changed <- FALSE
    }
}

# Then we update the centers of each cluster under the new assignment.
for (k in 1:groupNo){
        # Obtain all the points in the cluster k
        pointsIn <- dataset[cluster==k,]
        pointsMat <- as.matrix(pointsIn)
        # If there is at least one points in the cluster k, we update the center
        ## of the cluster k; otherwise, the center remains unchanged.
        if (nrow(pointsMat)>0){
            center[k,] <- colMeans(pointsMat)
        }else{
            center[k,] <- center[k,]
        }
     }
 }
# return the clustering result
return(cluster)
}
```

Now we shall use the function clusterByKmeans() to cluster the wine data and the iris data.

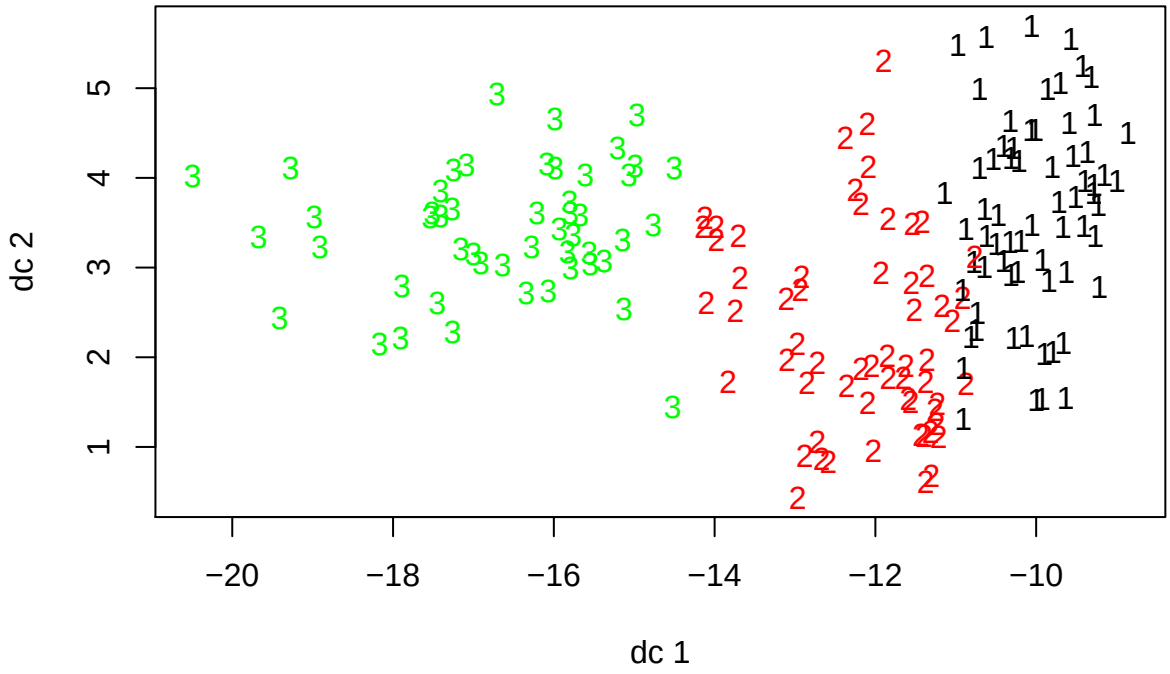
K means for the wine data without scaling:

```
# Obtain the wine data from the package "rattle"
data(wine, package="rattle")
head(wine)
```

```
##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids
## 1    1   14.23  1.71 2.43       15.6       127    2.80       3.06
## 2    1   13.20  1.78 2.14       11.2       100    2.65       2.76
## 3    1   13.16  2.36 2.67       18.6       101    2.80       3.24
## 4    1   14.37  1.95 2.50       16.8       113    3.85       3.49
## 5    1   13.24  2.59 2.87       21.0       118    2.80       2.69
## 6    1   14.20  1.76 2.45       15.2       112    3.27       3.39
##   Nonflavanoids Proanthocyanins Color  Hue Dilution Proline
## 1          0.28            2.29  5.64 1.04     3.92    1065
## 2          0.26            1.28  4.38 1.05     3.40    1050
## 3          0.30            2.81  5.68 1.03     3.17    1185
## 4          0.24            2.18  7.80 0.86     3.45    1480
## 5          0.39            1.82  4.32 1.04     2.93     735
## 6          0.34            1.97  6.75 1.05     2.85    1450
```

```
# Exclude the "Type" variable from the data inputs
# and assign the others to data.train
```

```
data.train <- wine[-1]
# Use set seed() so that the clustering results can be reproducible
set.seed(2475)
# Use k-means method to cluster the wines into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



As can be seen from the above plot, there exist some overlaps between clusters, for instance, between cluster 1 and cluster 2. The data points are indeed separated by the K means but it may not be very suitable to say that the data are well-separated.

We can compare the clustering results by k-means and the original classification of the 178 data indicated by the variable "Type":

```
# A comparison of the clustering results by k-means
# and the original classification indicated by "Type"
wt.km <- table(wine$Type, fit.km)
wt.km
```

```
##    fit.km
##      1  2  3
##   1  0 11 48
##   2 50 19  2
##   3 17 31  0
```

```
# Calculate the adjusted rand index, which is used for
## quantifying to what extent the two ways of clustering agree with each other.
randIndex(wt.km)
```

```
##       ARI
## 0.3986227
```

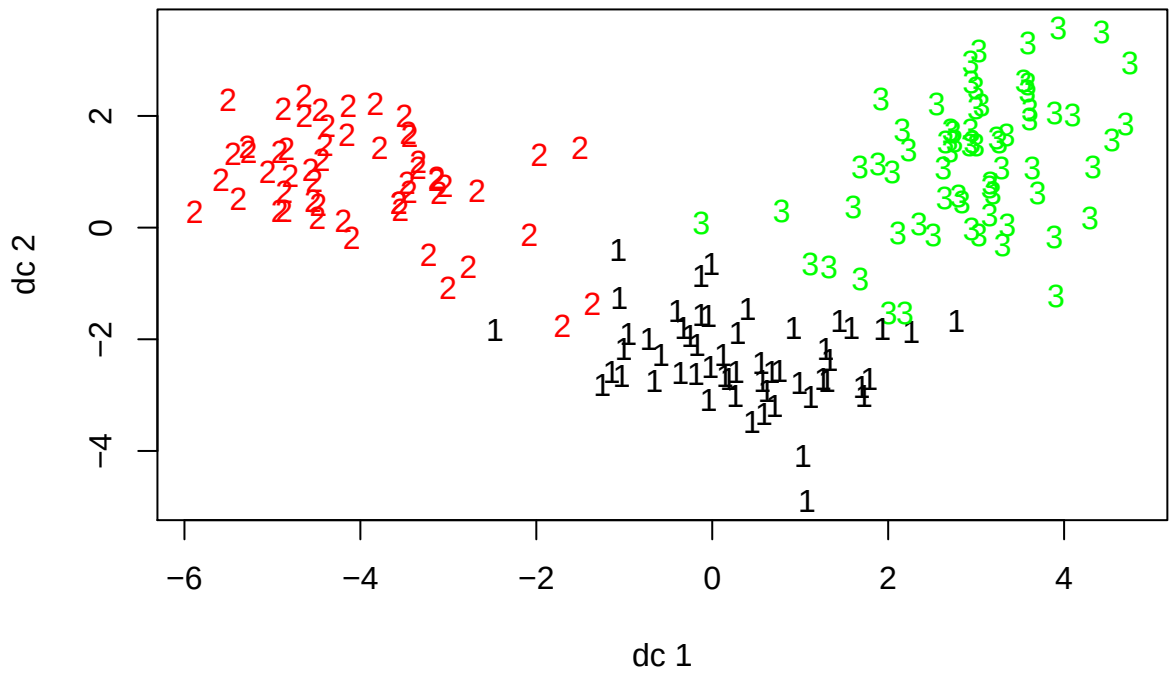
The above table shows directly that differences between the two partitions indeed exist. The value of the adjusted rand index is 0.3986227, which is not close to 1. Note that the adjusted rand index which is a measure of the degree of agreement between two partitions takes values between -1 (implying no agreement) and 1 (implying perfect agreement). Hence, we conclude that the partition obtained by K means does not match the original types of the wine well.

Now we repeat the above exercise using scaled data:

```
# Obtain the wine data from the package "rattle"
data(wine, package="rattle")
head(wine)
```

```
##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids
## 1    1   14.23  1.71 2.43       15.6       127    2.80       3.06
## 2    1   13.20  1.78 2.14       11.2       100    2.65       2.76
## 3    1   13.16  2.36 2.67       18.6       101    2.80       3.24
## 4    1   14.37  1.95 2.50       16.8       113    3.85       3.49
## 5    1   13.24  2.59 2.87       21.0       118    2.80       2.69
## 6    1   14.20  1.76 2.45       15.2       112    3.27       3.39
##   Nonflavanoids Proanthocyanins Color  Hue Dilution Proline
## 1          0.28            2.29  5.64 1.04     3.92    1065
## 2          0.26            1.28  4.38 1.05     3.40    1050
## 3          0.30            2.81  5.68 1.03     3.17    1185
## 4          0.24            2.18  7.80 0.86     3.45    1480
## 5          0.39            1.82  4.32 1.04     2.93     735
## 6          0.34            1.97  6.75 1.05     2.85    1450
```

```
# Exclude the "Type" variable from the data inputs,
# center and scale the rest of the data and assign the rest to data.train
data.train <- scale(wine[-1])
# Use set seed() so that the clustering results can be reproducible.
set.seed(2475)
# Use k-means method to cluster the wines into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



The command, data.train <- scale(wine[-1]), is used for center and scale the wine data excluding the variable Type. More specifically, for each column of wine[-1], first substract the corresponding column mean and then dividing that centered column by its standard deviation. In the data.train here, each column has a mean of zero.

Using the scaled data, we obtain a clustering result which looks better than what we obtained in the unscaled case, though minor overlaps still exist.

We then compare the clustering results by k-means and the original classification of the 178 data indicated by the variable "Type":

```
# A comparison of the clustering results by k-means
# and the original claassification indicated by "Type"
wt.km <- table(wine$Type, fit.km)
wt.km
```

```
##    fit.km
##      1  2  3
##   1  0  0 59
##   2 55  7  9
##   3  0 48  0
```

```
# Calculate the adjusted rand index
randIndex(wt.km)
```

```
##       ARI
## 0.7423375
```

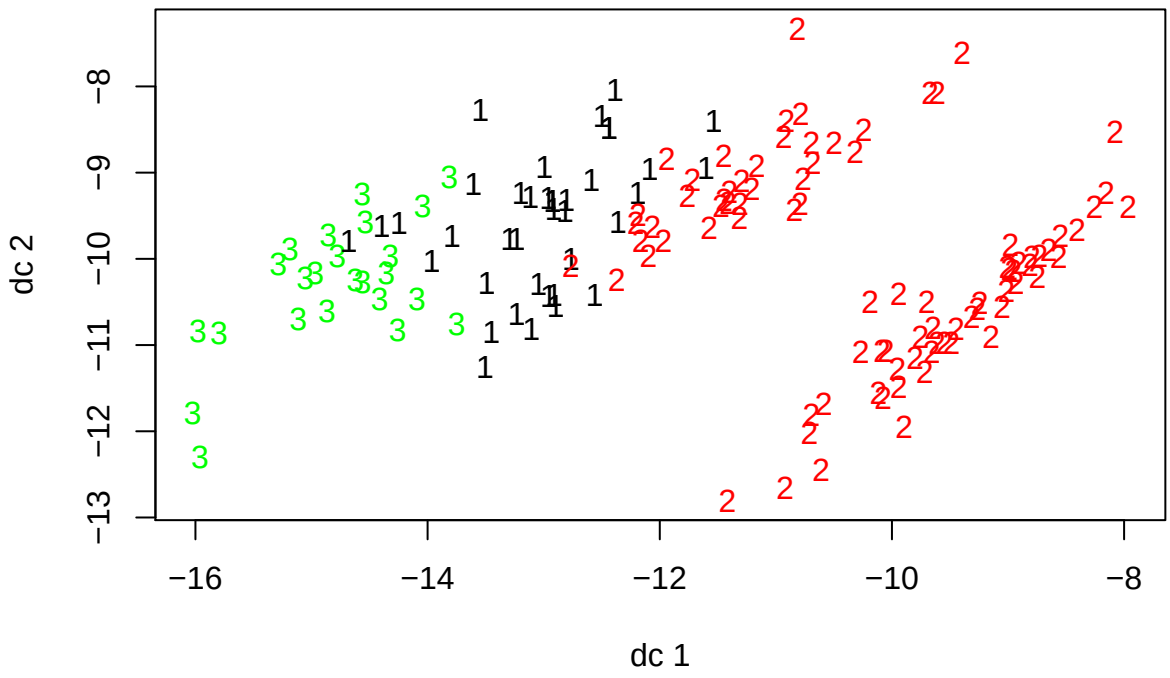
Now the table shows that there are fewer missing samples than in the unscaled case. The value of the adjusted rand index is 0.7423375, which is much larger than in the unscaled case. Hence, scaling helps improve the performance of K means in the sense that after scaling the clustering result using K means is more similar to the original partition.

In the following we show repeat the above exercise for the iris dataset. We first use the original dataset which is not scaled.

```
# Obtain the iris data
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# Exclude the "Species" variable and assign the rest to data.train
data.train <- iris[-5]
# Use set seed() so that the clustering results can be reproducible
set.seed(3000)
# Use k-means method to cluster the data into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



The plot here shows some overlaps between clusters, for instance, between cluster 1 and cluster 3 as well as between cluster 1 and cluster 2. It seems that K means does not separate the data points very well in this case.

Now we compare the clustering results by k-means and the original classification of the data indicated by the variable "species":

```
fs.km <- table(iris$Species,fit.km)
fs.km
```

```
##             fit.km
##               1  2  3
##   setosa      0 50  0
##   versicolor 14 36  0
##   virginica  25  1 24
```

```
randIndex(fs.km)
```

```
##       ARI
## 0.3609973
```

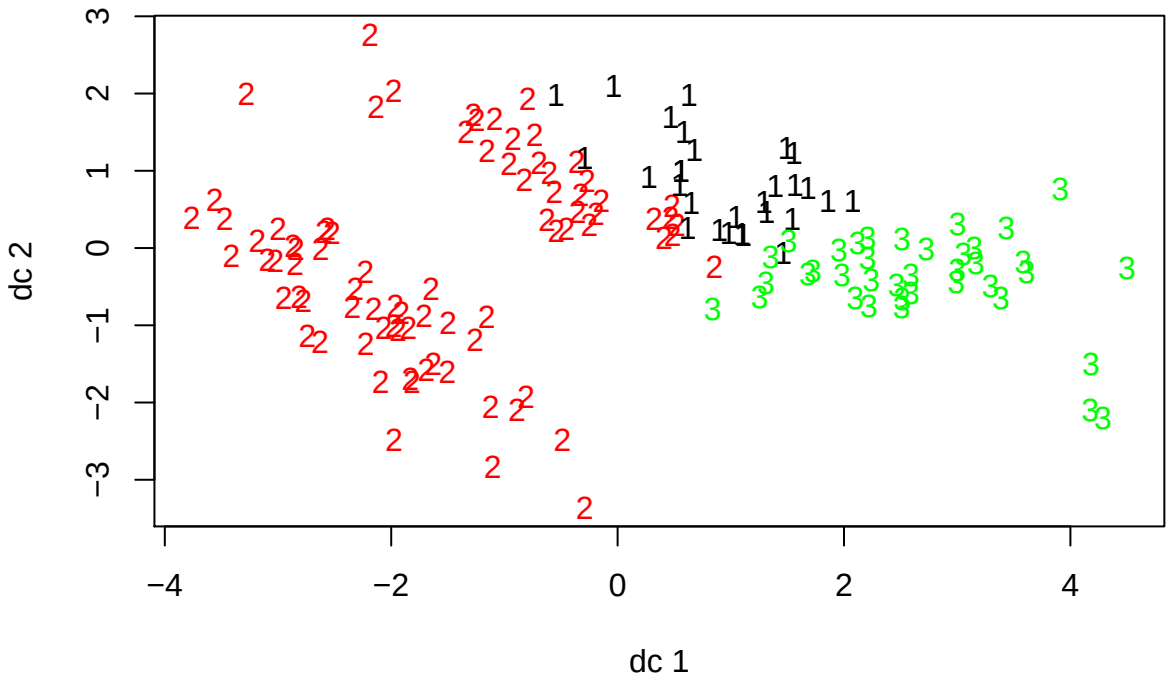
The adjusted rand index here is 0.3609973, which is not close to 1. So the algorithm's clusters do not match the original partition very well.

Now we instead use the scaled iris dataset:

```r
# Obtain the iris data
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
# Exclude the "Species" variable and assign the rest to data.train
data.train <- scale(iris[-5])
# Use set seed() so that the clustering results can be reproducible
set.seed(3000)
# Use k-means method to cluster the data into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



As shown in the plot, there are still some overlaps clusters, for instance, between cluster 1 and cluster 2 as well as between cluster 1 and cluster 3. K-means here still cannot separate the data points very well.

To quantify the comparison, we can still apply randIndex()

```
fs.km <- table(iris$Species,fit.km)
fs.km
```

```
##             fit.km
##               1  2  3
##   setosa      0 50  0
##   versicolor 10 32  8
##   virginica  19  1 30
```

```
randIndex(fs.km)
```

```
##       ARI
## 0.3510718
```

The adjusted rand index now is 0.3510718, which is even smaller than the unscaled case. In other words, for the iris dataset, scaling does not help improving the effectiveness of the k-means and K-means method in this case works better for the unscaled case.

As a summary, scaling is useful for improving the performance of K means for the wine data but not for the iris data.

Remark: when performing the K-means algorithm, we observe that the performance of this algorithm depends on the initial choice of group centers. Nevertheless, we choose the set seeds in this part so as to make the output reproducible.